Lesson 21:
THE ETHEREUM BLOCKCHAIN
29/05/2018
ETHEREUM TIMELINE

- Ethereum launched the system in May 2015

- main sources of information
  - the Ethereum White Paper [Butlerin 13]

- continuous updates from the development team posted on
  - the Ethereum blog
  - details on technical aspects and discussions available on GitHub

- Several releases:
  - Frontier: July 2015
  - **Homestead: March 2016**
  - Metropolis
  - Serenity
MERKLE TREES

Root Hash
\[ H = h(H0 + H1) \]

\[ H0 = h(H00 + H01) \]
\[ H00 = h(Value_0) \]
Value_0

\[ H01 = h(Value_1) \]
Value_1

\[ H10 = h(Value_0) \]
Value_2

\[ H11 = h(Value_3) \]
Value_3

\[ H01 = h(H010 + H011) \]

\[ H010 = h(Value_0) \]
Value_2

\[ H011 = h(Value_3) \]
Value_3
MERKLE PATRICIA TREES

- a data structure that is used extensively in Ethereum.
- combines the functionality of a classic Merkle and a Patricia trees
  - the same properties for integrity checking of data stored in the structure obtained with the Merkle tree,
  - optimizations for storage, insertion, and deletion operations obtained from the Patricia tree.
- all data added to the Merkle Patricia Tree is hashed and indexed in the format [key, value].
  - the key is the hash of the data
  - the value is the actual data.
- the keys become the traversal keys, while the data values are stored in the leaves of the tree.
MERKLE PATRICIA TREES

Data to be stored

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10102</td>
<td>Cat</td>
</tr>
<tr>
<td>010102</td>
<td>Dog</td>
</tr>
<tr>
<td>01010255</td>
<td>Mouse</td>
</tr>
<tr>
<td>010102D745</td>
<td>Fish</td>
</tr>
</tbody>
</table>

Root node

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>010102</td>
<td>H(Branch0)</td>
</tr>
</tbody>
</table>

Branch0

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>745</td>
<td>Fish</td>
</tr>
</tbody>
</table>

Branch1

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Mouse</td>
</tr>
</tbody>
</table>

key = H(value)
ETHASH: ETHEREUM MINING

- The Proof of Work mining algorithm for Ethereum is called Ethash.

- similar to Nakamoto Consensus, but more complex than the simple "hash twice" of Bitcoin to be ASIC resistant
  - make it difficult, or at least unprofitable to produce ASICs to increase the level of mining performance

- ASIC resistance:
  - “The algorithm for finding a valid proof of work includes, not only changing a nonce value but also fetching pieces of data from the DAG for each try in a pseudo-random way. Every 30000 blocks this data set is recalculated”
  - the most costly operation is not the hashing function, but rather the input/output operation of reading from memory
  - most regular personal computers are already optimized for I/O operations
  - it should be increasingly hard or expensive to create hardware that would speed up the mining process considerably
ETHASH: ETHEREUM MINING

\[(m, n) = \text{PoW}(H_{m}, H_{n}, d)\]

- mixHash
- Block Header without mixHash and nonce.
- nonce
- DAG
- Block Header.
THE ETHEREUM CONSENSUS

• a variation of the Nakamoto consensus

• the block selection strategy
  • no longer the longest chain wins, but rather the heaviest sub-tree wins

• heaviest sub-tree
  • is the tree that has the most combined computational work performed, not only on the main chain, but including valid computational work performed on closely related forks as well

• how to select the chain to link a block?
  • from a given block height, if there occurs a split in the blockchain the sub-tree with the most blocks generated is the one that should be selected
  • not just the longest chain.
THE ETHEREUM CONSENSUS

- the green boxes denote two separate subtrees A and B, of depth 4 and 3
- according to the Nakamoto consensus protocol
  - subtree A would be the valid chain, the longest chain.
- the purple box is an ommer block e.g. a stale block.
- ommer: gender neutral term for an aunt or an uncle
THE ETHEREUM CONSENSUS

- when the miner has to decide the block to be mining from, it considers the *heaviest subtree*
  - one of the lowest blocks in subtree B, B3 or B3'
- depending on which of B3 or B3' gets a child first
  - one of them will become an ommer as new blocks are mined.
- a limit to the depth of the heaviest-sub tree
  - a maximum of 6 ancestors,
  - a maximum of 2 ommers included in each block.
THE ETHEREUM CONSENSUS

• Ommer (uncle): a block whose parent is equal to the current block’s parent’s parent.

• Block times in Ethereum are around 15 sec. This is much lower than that in Bitcoin (10 min).

• This enables faster transaction. But there are more competing blocks, hence a higher number of orphaned blocks.

• The purpose of ommers is to help reward miners for including these orphaned blocks.

• A fraction of the reward obtained by normal minimh.
ETHEREUM CONSENSUS: PROOF OF STAKE

• a category of consensus algorithms for public blockchains that depend on a validator's economic stake in the network.

• instead of using computation as the limiting economic resource, use digital assets inside the system bitcoin, ether, or whatever other coin inside the platform

• proving ownership of digital assets is easy: just sign a digital signature

• Casper solution

• different versions of the algorithm.
**Ethereum Block**

**Block header**

- **parentHash**
  
  [256-bit] Hash of parent block header.

- **ommersHash**
  
  [256-bit] Hash of ommers list.

- **beneficiary**
  
  [160-bit] Address of miners account.

- **stateRoot**
  
  [256-bit] Hash of root of State Tree.

- **transactionsRoot**
  

- **receiptsRoot**
  

- **logsBloom**
  
  Bloom filter for Logs of the block.

- **difficulty**
  
  Difficulty level of this block.

- **number**
  
  Value equal to number of ancestor blocks.

- **gasLimit**
  
  Current level of gas expenditure for this block.

- **gasUsed**
  
  Total gas used in the transactions in this block.

- **timestamp**
  
  Reasonable creation time of block.

- **extraData**
  
  [≥256-bit] arbitrary byte array.

- **mixHash**
  
  [256-bit] Proves Proof of Work combined with nonce.

- **nonce**
  
  [64-bit] Proves Proof of Work combined with mixHash.

**Transaction List**

List of all transactions included in this block.

**Ommers List**

List of all Ommers referenced in block.